CMPSC 465 Data Structures & Algorithms
Spring 2024 Mehrdad Mahdavi and David Koslicki

Worksheet 10

Wednesday, Mar 27, 2024

1. **Proof Technique Review.** Please review the following common proof techniques and highlight how they are used in this week's problems:

   (a) Direct Proof (Problem 3)

   (b) Proof by Contradiction (Problem 2)

   (c) Proof by Contrapositive (No problem, just know that $a \to b$ is equivalent to $\neg b \to \neg a$)

   (d) Proof by Induction (Problem 6)

2. **Prefix-Free Encoding.** Given a finite alphabet Γ, a prefix-free encoding assigns each symbol in Γ a binary codeword such that no codeword is a prefix of another codeword. A prefix-free encoding is minimal if it is not possible to arrive at another prefix-free encoding by reducing any of the codewords. For example, the encoding $0, 101$ is not minimal because the codeword 101 can be reduced to 1, and the encoding would still be prefix-free.

   Show that any minimal prefix-free encoding can be represented by a full binary tree in which each leaf corresponds to a unique element of Γ, and each codeword can be generated by tracing the path from the root to that symbol's leaf.

   Solution:

   If we consider the binary tree with all strings of length k at level k and the left and right branches representing adding a 0 and 1 respectively, it contains all the binary strings and hence all the strings in the encoding (we only need to have number of levels equal to maximum length of a string). Also, since all intermediate nodes in a path from the root to a node v are prefixes of v, the strings of the encoding must be all at leaves since it is prefix-free.

   To argue that the tree must be full, suppose for contradiction that a node u corresponding to a string s has only one child v corresponding to some string s'. Since a codeword is a leaf in the subtree rooted at u iff it has s, and hence also s' as a prefix, replacing s' by s in all these codewords gives a better encoding. However, this is a contradiction since we assumed our encoding to be minimal.

3. **Service scheduling.** A server has n customers waiting to be served. Customer i requires t_i minutes to be served. If, for example, the customers were served in the order $t_1, t_2, t_3, \ldots$, then the ith customer would wait for $t_1 + t_2 + \cdots + t_i$ minutes.

   We want to minimize the total waiting time

$$T = \sum_{i=1}^{n} (\text{time spent waiting by customer } i)$$

Given the list of t_i, give an efficient algorithm for computing the optimal order in which to process the customers.

Solution:

We simply proceed by a greedy strategy, by sorting the customers in the increasing order of service times and servicing them in this order. The running time is $O(n \log n)$.

To prove the correctness, for any ordering of the customers, let $s(j)$ denote the jth customer in the ordering. Intuitively, customers with a low service time should be positioned near the front of the ordering so they aren't all waiting for the customers with longer service times. For any ordering, if $t_{s(i)} > t_{s(j)}$ for $i < j$, swapping the positions of the two customers gives a better ordering. Since we can generate all possible orderings by swaps, an ordering that has the property $t_{s(1)} \leq \ldots \leq t_{s(n)}$ must be the global optimum. This is exactly the ordering we output.

4. **Specified Leaves MST.** Given a connected, undirected graph $G = (V, E)$ with edge weights w_e and a subset of vertices $U \subset V$, give an $O(|V| + |E| \log |V|)$ algorithm to determine the lightest spanning tree T such that every $u \in U$ is a leaf node in T.

   Solution:

   We first note that each $u \in U$ must have at least one neighbor in $V \backslash U$, otherwise the problem has no solution. If T is the optimal tree, then $T \backslash U$ must be a spanning tree of $G \backslash U$. Moreover, it must be a minimum spanning tree since the nodes in U can be attached as leaves to *any* spanning tree. Hence, we first find an MST of $G \backslash U$ using Kruskal's algorithm in $O(|E| \log |V|)$ time. Then, for each $u \in U$, we add the lightest edge between u and $G \backslash U$ in $O(|V| + |E|)$ time.

5. **Longest Encoding.** Under a Huffman encoding of n ($n \geq 2$) symbols with frequencies $f_1, f_2, \ldots, f_n$ (assuming they are in non-increasing order), what is the longest length a codeword could possibly have? In what kind of case(s) (specifically, in terms of n and $f_1, f_2, \ldots, f_n$) can this happen? Show how you derive those case(s).

   Solution:

   From problem 2, we know that we can use a full binary tree to represent the encoding of each symbol with symbols as the leaves. So, this question is simply asking what the largest possible depth is for a full binary tree with n leaves. The answer is $n - 1$ which happens when only two leaves are at the lowest level.

   To achieve this, we need to make sure that the nodes being combined are a leaf and an internal node except for the first iteration, where both of them can only be leaves. This puts requirements on the frequencies that in the cases of $n \geq 4$, $f_{n-i-2} > \sum_{j=0}^{i} f_{n-j}$ for all i from 1 to $n-3$. Since $f_{n-i-2} \geq f_{n-i-1}$, f_{n-i-1} corresponds to a leaf and $\sum_{j=0}^{i} f_{n-j}$ corresponds to an internal node and they will be combined, which achieves what we want. In the cases where $n = 3$ or $n = 2$, no requirements are needed to achieve what we want.

   In summary, this can happen in any of the following cases:

   - $n = 2$
   - $n = 3$
   - $n \geq 4$ and $f_{n-i-2} > \sum_{j=0}^{i} f_{n-j}$ for all i from 1 to $n-3$

6. Fibonacci Frequencies.

(a) What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \qquad b:1 \qquad c:2 \qquad d:3 \qquad e:5 \qquad f:8 \qquad g:13 \qquad h:21$$

(b) Generalize your answer to give the optimal encoding when the frequencies are the first n Fibonacci numbers. Prove that your encoding is optimal by showing that it is produced by the Huffman encoding algorithm. (*Hint*: Consider relating Fibonacci number $k+1$ to the sum of the first $k-1$ Fibonacci numbers).

Solution:

(a) Optimal Huffman code is as follows: $a = (0000000), b = (0000001), c = (000001), d = (00001), e = (0001), f = (001), g = (01), h = (1)$

(b) This generalizes to having the first n Fibonacci numbers as the frequencies in that the $k < n^{th}$ most frequent letter has codeword ($0^{k-1}1$), and the n^{th} most frequent letter having codeword (0^{n-1}). To see that this holds, we will prove the recurrence:

$$\sum_{i=0}^{n-1} F(i) = F(n+1) - 1 \tag{1}$$

This will show that we should join together the letter with frequency $F(n)$ with the result of joining together the letters with smaller frequencies. We will prove it by induction. For $n = 1$ is is trivial to check. Now, suppose that we have $n-1 \geq 1$, then,

$$F(n+1) - 1 = F(n) + F(n-1) - 1 = F(n-1) + \sum_{i=0}^{n-2} F(i) = \sum_{i=0}^{n-1} F(i) \tag{2}$$

To use this fact, to show the desired Huffman code is optimal, we claim that as we are greedily combining nodes, that at each stage we can maintain one node which contains all of the least frequent letters. Initially, this consists of just the least frequent letter. Then, at stage k, inductively, we assume that it contains the k least frequent letters. This means that it has weight $\sum_{i=0}^{k-1} F(i) = F(k+1) - 1$. All of the other nodes at this stage have weights $\{F(k), F(k+1), ...F(n-1)\}$. So, the two lowest weight nodes are this node containing the k least frequent letters and the node containing the $k+1$-st least frequent letter. Therefore, the greedy algorithm tells us that we should combine those nodes leaving us with a node containing the $k+1$ least frequent letters for stage $k+1$, completing the induction. Of course, the code that we have given is not uniquely optimal, because left and right children (represented by 0 and 1 respectively) are an artificial choice, we could assume that at each stage we are adding in the singleton as the 1 child of the new parent, getting us our code.